Web technologies

PHP Advanced Topics

Lecture slides – Winter term 2010
Alpen-Adria-University Klagenfurt

Christian Koncilia

Object Oriented PHP

OO PHP

- PHP is an object-oriented language
- OO support basically as in Java
  - Classes, Methods, Objects
  - Visibility: private, public, protected
  - Static classes and methods
  - Inheritance
  - No multiple inheritance
  - Semi-multiple inheritance using interfaces

PHP My First Class

php_simpleclass.php

```
<?php
class myClass {
   private $i;
   public function __construct($i = 0) {
      $this->i = $i;
   }
   public function getI() {
      return $this->i;
   }
}

$c = new myClass();
// echo $c->i . "<br>";  // would result in a fatal error
echo $c->getI() . "<br>";
$d = new myClass(5);
echo $d->getI() . "<br>";
?>
```

PHP Inheritance

php_persons.php

```
class Person {
   var $firstname;
   var $lastname;
}

class Student extends Person {
   var $matrikelnr;
}

$koncilia = new Person();
$smith = new Student();
$smith->firstname = "Tim";
$smith->lastname = "Smith";
$smith->matrikelnr = 1025082;
echo $smith->firstname . " " . $smith->lastname . " " .
     $smith->matrikelnr;
```

PHP Const / self / this

php_selfthis.php

```
class MalePerson {
    const sex = "m";
    var $firstname;
    var $lastname;
    public function printPerson() {
        echo $this->firstname . "<br>";
        echo $this->lastname . "<br>";
        echo self::sex . "<br><br>";
    }
}

$koncilia = new MalePerson();
$koncilia->firstname = "Christian"; $koncilia->lastname = "Koncilia";
$koncilia->printPerson();
echo '$koncilia->sex: ' . $koncilia::sex;
```

PHP Bad Programming Style

- Forget everything you saw until now! Immediately!
- Bad programming style:
  - Everything public
  - No getters/setters
  - Everything in one file
- Better way:
  - Each class in a single file
  - Require / Include / Autoload when needed

PHP Person Class redefined

Person.php
Filename doesn't have to be Classname. However, it makes things easier (see autoloading)

```
<?php
class Person {
   private $firstname;
   private $lastname;

   public function __construct($f = null, $l = null) {
           $this->setFirstname($f);
           $this->setLastname($l);
   }

   public function getFirstname() {return $this->firstname;}
   public function getLastname() {return $this->lastname;}
   public function setFirstname($f) {$this->firstname = $f;}
   public function setLastname($l) {$this->lastname = $l;}
}

// why is there no closing PHP-tag? Actually, if your file consists ONLY
// of PHP code, it is a good idea to ommit the closing tag in order to
// avoid accidental injection of trailing whitespace in the output (in
// the generated HTML)
```

PHP Student Class redefined

Student.php

```
<?php

require_once "Person.php";

class Student extends Person {
   private $matrikelnr;

   public function __construct($f = null, $l = null, $m = null) {
            parent::__construct($f, $l);
            $this->setMatrikelnr($m);
   }
   public function getMatrikelnr() {return $this->matrikelnr;}
   public function setMatrikelnr($m) {$this->matrikelnr = $m;}
}
```

PHP Using the classes

php_better.php

```
<?php
require_once "Person.php";
require_once "Student.php";

$koncilia = new Person("Christian", "Koncilia");
$smith = new Student("Tim", "Smith", "939282");

echo $koncilia->getFirstname() . " " . $koncilia->getLastname() . "<br>";

echo $smith->getFirstname() . " " . $smith->getLastname() . " " .
     $smith->getMatrikelnr();
?>
```

PHP Autoload

php_autoload.php

- It‘s annoying to manually load all classes if you are using many classes in one file…
- Use autoload!

```
<?php
   function __autoload($class_name) {
      require_once $class_name . '.php';
   }

   $koncilia = new Person("Christian", "Koncilia");
   $smith = new Student("Tim", "Smith", "939282");
   echo $koncilia->getFirstname() . " " . $koncilia->getLastname() . "<br>";
   echo $smith->getFirstname() . " " . $smith->getLastname() . " " .
        $smith->getMatrikelnr();
?>
```

PHP Overloading

Person2.php

Student2.php

php_overloading.php

```
<?php
class Person2 {
    … as seen before…
    public function printMe() {
        echo $this->getFirstname() . " " . $this->getLastname();
    }
}
```

```
<?php
class Student2 extends Person2 {
    … as seen before…
    public function printMe() {
        parent::printMe();
        echo " " . $this->getMatrikelnr();
    }
}
```

PHP Magic Methods

php_magicmethods.php

```
class myClass {
   private $i;
   private $j;

   public function __set($name, $value) {
      $this->$name = $value;
   }

   public function __get($name) {
      return $this->$name;
   }
}

$c = new myClass();
$c->i = 5;
echo $c->i;
$c->x = " is less than 6";
echo $c->x;
```

PHP & Databases

PHP and Databases

- Abstract layers
  - DBA
  - DBX
  - ODBC
  - PDO
- Database vendor specific layers
  - MySQL
  - Oracle
  - MS SQL
  - DB2
  - ….

PHP PDO

php_db1.php

```
<?php
   // run DATABASE.sql before you start this program

   $pdo = new PDO(
          'mysql:host=localhost;dbname=dbtech',
          'root',  // username
          'root',  // password
          array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8")
   );
   $stmt = $pdo->prepare("SELECT * FROM products");
   $stmt->execute();
   $dbresult = $stmt->fetchAll(PDO::FETCH_ASSOC);  // [['attr_name' => 'value']]
   foreach ($dbresult as $record) {
      foreach ($record as $k => $v) {
         echo $k . ": " . $v . "<br>";
      }
   }
?>
```

PHP Fancy Table

php_db2.php

```
…
$stmt = $pdo->prepare("SELECT * FROM products");
$stmt->execute();
$dbresult = $stmt->fetchAll(PDO::FETCH_ASSOC);

echo "<table wdith=\"100%\">\n";
echo "<tr>\n";
// add the table headers
foreach ($dbresult[0] as $key => $useless){echo "<th>$key</th>";}
echo "</tr>";
// display data
foreach ($dbresult as $row){
   echo "<tr>";
   foreach ($row as $key => $val){
      echo "<td>$val</td>";
   }
   echo "</tr>\n";
}
echo "</table>\n";
```

HTML/CSS Excursion:

Use tables only when you need tables!

Otherwise use <div>s + float + clear

PHP Prepare Statement

php_dbprepare.php

```
…
$pdo = new PDO(
   'mysql:host=localhost;dbname=dbtech',
   'root',  // username
   'root',  // password
   array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8")
);
$stmt = $pdo->prepare("SELECT * FROM products where name = :name");
$stmt->bindParam(':name', $lookingforname, PDO::PARAM_INT);
$lookingforname = 'Banana';
$stmt->execute();
$dbresult = $stmt->fetchAll(PDO::FETCH_ASSOC);
…
```

PHP CRUD

- Basically, all CRUD (Create, Read, Update, Delete) Operations look the same:

1. Use prepare statement to define the query
2. Bind variables
3. Set values for variables
4. Execute statement

PHP cruD

php_crud.php

```
try {
   $stmt = $pdo->prepare("DELETE FROM products WHERE name = :prodname");
   $stmt->bindParam(':prodname', $bananas, PDO::PARAM_STR);
   $bananas = 'Banana';

   if ($stmt->execute()) {
      echo "Great! Deleting the bananas worked!";
   } else {
      throw new Exception("There is something wrong with your query");
   }
} catch (Exception  $e) {
   echo "Sorry, didn't really work... Error: " . $e->getMessage();
}
```

PHP Crud

php_crud2.php

```
try {
   $stmt = $pdo->prepare("INSERT INTO `products` (`name`,`price`,`available`,`release`)
                          VALUES (:name,:price,:available,:release)");

   $stmt->bindParam(':name', $banana_name);
   $stmt->bindParam(':price', $banana_price, PDO::PARAM_INT);
   $stmt->bindParam(':available', $banana_available, PDO::PARAM_INT);
   $stmt->bindParam(':release', $banana_release, PDO::PARAM_STR);

   $banana_name = 'Banana'; $banana_price = 2; $banana_available = true;
   $banana_release = '1841-12-24 00:00:00';
   $count = $stmt->execute();
   echo "Great! Inserted a brand new banana!
         Database says that I inserted $count banana...";
} catch (Exception $e) {
   echo "Sorry, didn't really work... Error" . $e->getMessage();
}
```

PHP crUd

php_crud3.php

```
try {
   $stmt = $pdo->prepare("UPDATE products SET price = :newprice WHERE name = :name");
   $stmt->bindParam(':name', $banana_name, PDO::PARAM_STR);
   $stmt->bindParam(':newprice', $banana_price, PDO::PARAM_INT);

   $banana_name = 'Banana';
   $banana_price = 2002;

   $stmt->execute();
   echo "Great! Updated the banana price!";
} catch (Exception $e) {
   echo "Sorry, didn't really work... Error" . $e->getMessage();
}
```

PHP Query Errors

php_dberrors.php

- Watch out! Pitfall!
- Default for PDO error reporting is `PDO::ERRMODE_SILENT`
- This means: errors in your query will NOT be reported!
- Why: security issues (reporting native error messages to the user is always risky)
- See php_dberrors.php

File operations

PHP Reading a File

php_file.php

```
$lines = file('php_file.php');
foreach ($lines as $line_num => $line) {
        echo "Line #<b>{$line_num}</b> : " . htmlspecialchars($line) . "<br>\n";
}
```

See next example

- "`file_get_contents('php_file.php')`" exactly like "`file(…)`" except that it returns the content in a string (not an array)

PHP Reading from an URL

php_fileurl.php

```
$lines = file('http://www.google.com');
foreach ($lines as $line_num => $line) {
         echo "Line #<b>{$line_num}</b> : " . htmlspecialchars($line) . "<br>\n";
}
```

- Try what happens if you substitute "`htmlspecialchars($line)`" with "`$line`"

PHP Byte by Byte

php_filestep.php

```
$f=fopen('php_filestep.php','r');
$size = 1; // read byte by byte
while(!feof($f)) {
          echo fread($f,$size) . "<br>";
}
fclose($f);
```

PHP Line by Line

php_filestep2.php

```
$file = fopen ("php_filestep2.php", "r");
while (!feof($file)) {
          $buffer = fgets($file);
          echo htmlspecialchars($buffer) . "<br>";
}
```